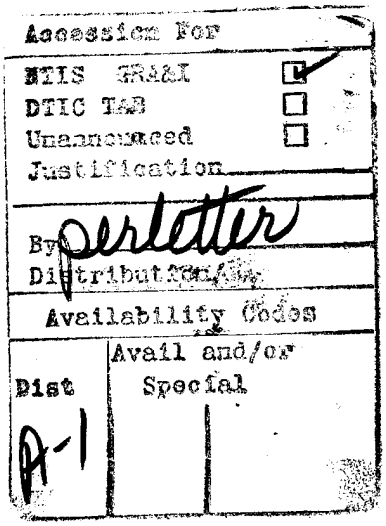
ANNUAL REPORT FY1994

Flexible Reactive Control for Multi-Agent Robotic Systems in Hostile Environments

ONR/ARPA Grant #N00014-94-1-0215

Prepared by: Ronald C. Arkin (P.I.), Jonathan Cameron,
Doug MacKenzie, Tucker Balch, and Khaled Ali
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
email: arkin@cc.gatech.edu
Fax: (404) 853-9376
Phone: (404) 894-8209

Contents



1. Introduction

This document constitutes the 1994 Annual Report for the ONR/ARPA Grant #N00014-94-1-0215 entitled Flexible Reactive Control for Multi-Agent Robotic Systems in Hostile Environments. This project is supported by ARPA's Real-time Planning and Control Program and has as a customer ARPA's UGV Demo II program. This first annual report reflects the first year's accomplishments within the context of an overall three year research program.

The goals of this research are to produce intelligent, flexible, reactive behaviors and methods for specifying and communicating information between multiagent teams. In particular we have been studying three closely related subjects:

- Formation Control - to allow teams of robotic agents to move in a coordinated manner through a potentially hostile environment without interfering with other active navigational behaviors.
- Teleautonomous Control of Multi-agent Teams - to allow a massive reduction in cognitive workload for the control of a group of robotic vehicles by permitting commands to be specified at the team level rather than at the individual agent level.
- Team Mission Specification Methods - to provide robust and flexible mission specification for reactive team military scenarios.

It is intended that all of these projects will be fielded in some capacity for the upcoming ARPA Demo C and Demo II scenarios involving 2-4 HUMMVs. This report surveys the progress made to date in each of these areas.

2. Formation Control

This research concerns the development of behaviors for formation maintenance in heterogeneous societies of mobile robots. The target platform is a set of four robotic vehicles to be employed as a scout unit by the U.S. Army. Formations are important in this application as they allow the unit to utilize its sensor assets more efficiently than if the robots are arranged randomly.

The robots in this work are heterogeneous in that each is assigned a unique identification number (ID). A robot's position in formation depends upon its ID. There are no other differences between robots. In future work the robots may differ substantially in their sensor capabilities, making it appropriate for specific robots to occupy particular positions in a formation.

Formation control is one part of a more complex behavioral assemblage which includes other components for task achievement. In addition to maintaining their position in formation, robots must simultaneously move to a goal location and avoid obstacles.

The formation behaviors presented here are implemented as *motor schemas*. Readers not familiar with motor schema-based reactive control are referred to [4].

2.1 Simulation Environment

Results were generated in simulation on a Sun SPARC IPC under SunOS and the X11 graphical windowing system. The simulation environment is a 1000 by 800 meter two dimensional "playing field" upon which various sizes and distributions of circular obstacles may be scattered. Each simulated robot is a separately running C program that interacts with the simulation via a Unix socket. The simulation displays the environment graphically and maintains world state information which it transmits to the robots as they request it. Figure 1 shows a typical simulation run. The robots are displayed as five-sided polygons (rectangles with points), while the obstacles are black circles. The robots' paths are depicted with solid lines.

Sensors allow a robot to distinguish between three perceptual classes: robots, obstacles and goals. The robots' sensors and actuators are implemented in the main simulation. When one of the robot's perceptual processes requires obstacle information for example, a request for that data is sent via the socket to the simulation. The information returned is a list of angle and range data for each obstacle in sensor range. Robot and goal sensor information is provided similarly. The robot moves by transmitting its desired velocity to the simulation process. The simulation process automatically maintains the position and heading of each robot.

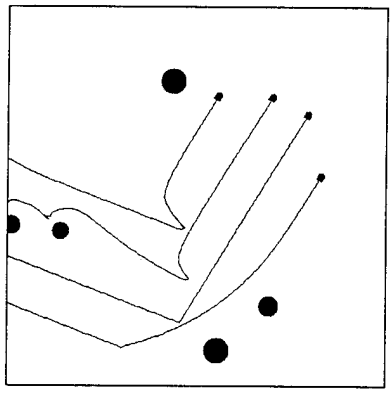


Figure 1: Typical simulation run showing four robots in a wedge formation executing a 90 degree left turn.

2.2 Formations

Several formations for four robots are considered:

- *line* - where the robots travel line-abreast.
- *column* - where the robots travel one after the other.
- *diamond* - where the robots travel in a diamond.
- *wedge* - where the robots travel in a "V".

For each formation, each robot has a specific position (based on it's ID). Figure 2.2 shows the formations and robots' positions within them.

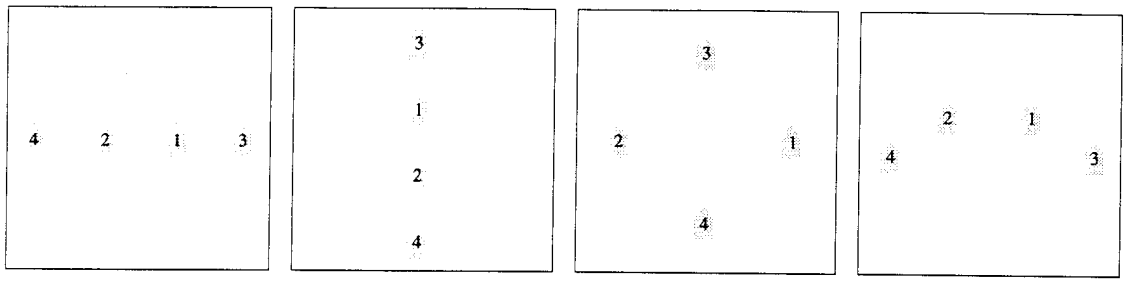


Figure 2: Formations for four robots (from left to right: line, column, diamond, wedge)

2.3 Behavioral Integration

The focus is on formation maintenance, but formation behaviors must meld with other behaviors such as obstacle avoidance. In the examples here, the task for each robot is to move to a goal location, avoid obstacles, avoid collisions with other robots and maintain formation position simultaneously. The primitive behaviors, $move-to-goal$, $avoid-static-obstacle$, $avoid-robot$, and $maintain-formation$, respectively, implement these behaviors. Each behavior generates a vector representing desired direction and magnitude of movement. A gain value indicates the relative importance of the individual behaviors. The high-level combined behavior is generated by multiplying the output of each primitive behavior by its gain, then summing the results. Gains and other schema parameters are listed in Table 1. See [4] for a more complete discussion of the parameters of the $avoid-static-obstacle$ and $move-to-goal$ motor schemas.

2.4 Approach

Formation maintenance is accomplished in two steps: first, a perceptual process, $detect-formation-position$, determines the robot's proper position in the formation; second, the motor schema $maintain-formation$ generates a movement vector towards it.

Parameter	Value	Units
avoid − static − obstacle		
gain	2.0	
sphere of influence	50	meters
minimum range	5	meters
avoid − robot		
gain	2.0	
sphere of influence	20	meters
minimum range	5	meters
move − to − goal		
gain	1.0	
maintain − formation		
gain	1.0	
desired spacing	50	meters
controlled zone radius	25	meters
dead zone radius	0	meters

Table 1: Parameter values used for motor schemas.

2.4.1 Formation Position References

Each robot must compute its proper position in the formation for each movement step. Three techniques for formation position determination are being explored:

- **Unit-center-referenced**: a unit-center is computed by averaging the x and y positions of each robot. Each robot determines its formation position relative to that center.
- **Leader-referenced**: each robot determines its formation position in relation to the navigational leader (the unit leader is not necessarily the navigational leader). The navigational leader does not attempt to maintain formation; the other robots are responsible for formation maintenance.
- **Neighbor-referenced**: each robot maintains a position relative to one other robot.

These relationships are depicted in Figure 3. Arrows show how the formation positions are determined. Each arrow points *from* a robot *to* the associated reference. The perceptual schema *detect−formation−position* must use one of these references to determine the proper position for the robot. The spacing between robots is determined by the *desired spacing* parameter of *detect − formation − position*.

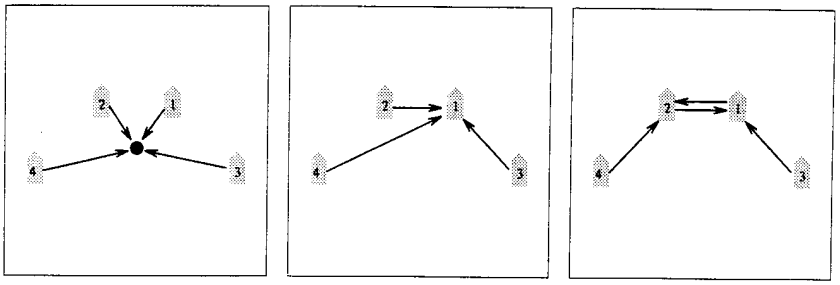


Figure 3: Formation position determined by various reference techniques (from left to right: unit-center, leader, neighbor)

2.4.2 The *maintain – formation* Motor Schema

Once the desired formation position is known, the *maintain – formation* motor schema generates a movement vector towards it. In all cases the direction of the vector is towards the desired formation position. The magnitude of the vector depends upon how far the robot is from the desired position. Figure 4 illustrates three zones (defined by distance from the desired position) used for magnitude computation. The radii of these zones are parameters of the *maintain – formation* schema. In the example case, robot 3 is attempting to maintain a formation position to the right of and behind robot 1. Robot 3 is in the controlled zone, so a moderate force towards the desired position (forward and right) is generated by *maintain – formation*. The magnitude of the vector is computed as follows:

- **Ballistic zone**: magnitude set at maximum (the schema's gain value).
- **Controlled zone**: magnitude varies linearly from maximum at the farthest edge of the zone to zero at the inner edge.
- **Dead zone**: in the dead zone vector magnitude is always zero.

2.5 Results

The *line*, *column*, *wedge* and *diamond* formations have been implemented using the unit-center-referenced and leader-referenced approaches. Neighbor-referenced formations are under development.

Figure 5 illustrates robots moving in each of the basic formations using the leader-referenced approach. In each of these simulation runs the robots were first initialized on the left side of the simulation environment, then directed to proceed to the lower center of the frame. After the formation was established, a 90 degree turn to the left was initiated. Results are similar for the unit-center-referenced formations.

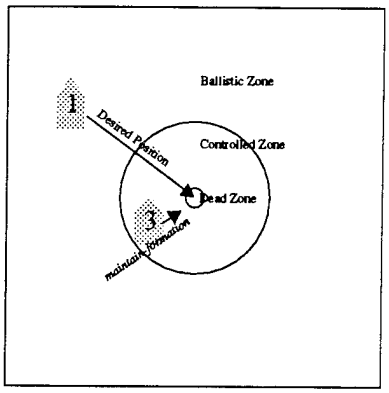


Figure 4: Zones for the computation of *maintain − formation* magnitude

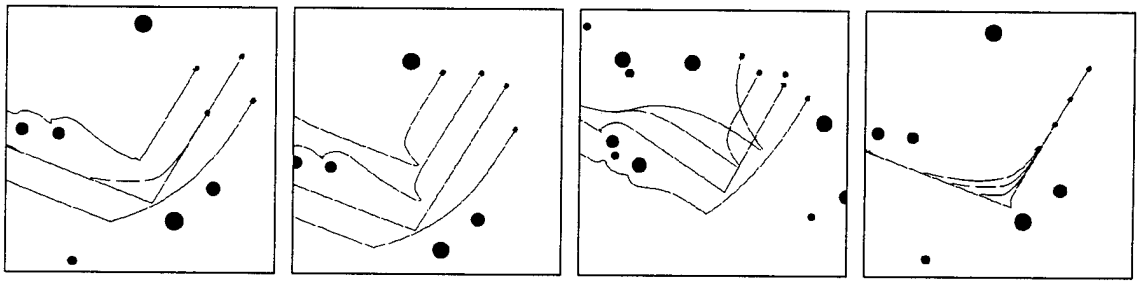


Figure 5: Four robots in leader-referenced *diamond*, *wedge*, *line* and *column* formations.

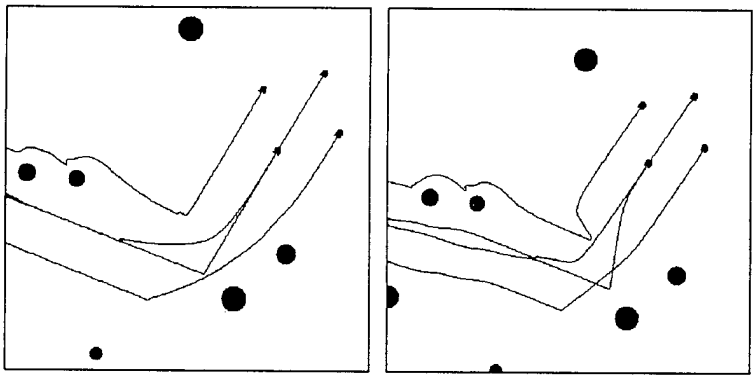


Figure 6: Comparison of leader-referenced (left) and unit-center-referenced (right) diamond formations.

Qualitative differences between the two approaches can be seen as a formation of robots moves around obstacles and through turns (see Figure 6). For leader-referenced formations any turn by the leader causes the entire formation to shift accordingly, but if another robot turns, the rest of the formation is not affected. In unit-center-referenced formations any robot move or turn impact the entire formation. In turns for leader-referenced formations, the leader simply heads in the new direction; other robots must adjust their position to move into formation. In unit-center-referenced turns, the whole formation initially appears to spin about a central point, as it aligns with the new heading.

2.6 Discussion

Which formation type and formation reference is best? The answer depends on many factors, including task, environment, and mission constraints. More simulations and analysis of qualitative results will help robot system designers match formations to applications. Sensor and mission constraints may also dictate specific types of formations. In any case, these applications probably rule out the use of unit-center-referenced formations:

- **Human Leader:** A human cannot reasonably compute a formation's unit-center on the fly, especially while simultaneously avoiding obstacles. A leader-referenced formation is most appropriate in this application.
- **Communications Restricted Applications:** The unit-center approach requires a transmitter and receiver for each robot and a protocol for exchanging position information. Conversely, the leader-referenced approach only requires one transmitter for the leader, and one receiver for each following robot. Bandwidth requirements are cut by 75% in a four robot formation.
- **Passive Sensors for Formation Maintenance:** Unit-center-referenced formations place a great demand on passive sensor systems (e.g. vision). In a four robot visual formation for instance, each robot would have to track three other robots which may spread across a 180 degree field of view. Leader- and neighbor-referenced formations only call for tracking one other robot [1].

2.7 Technical Transfer

The formation behaviors are being transferred to the UGV project at two levels: first as part of onboard software for UGV Demo C, and second as part of a military mission description and simulation package called MissionLab.

[1]The neighbor-referenced approach is used by U.S. Army Infantry Squads and Air Force fighters for visual formations.

At Demo C, formation routines will be installed on two UGVs to implement a "follow-the-leader" behavior during road following. At a Technical Demo to be held in conjunction with Demo C, we expect to additionally demonstrate a comprehensive suite of formation behaviors for two robots. Tech transfer for this application requires integration with the DAMN Arbiter, and the LinkManager software aboard the UGVs. Work towards this integration has already begun: interface descriptions are presently being worked out with Martin Marietta, and a version of the DAMN Arbiter is in use at Georgia Tech for integration testing.

In MissionLab, a military mission simulator, formation software has been combined with other behaviors to enable robots to carry out complex missions. Using the script-like language in MissionLab, a commander may specify complete missions for a team of robots then execute them in simulation. Section 4.3 of this report describes this in more detail.

3. Teleautonomy

Reactive multi-agent robotic societies can be potentially useful for a wide-range of tasks. This includes operations such as foraging and grazing (e.g., [1,11,7]) which have applicability in service (vacuuming and cleaning), industrial (assembly) and military (convoy and scouting) scenarios.

Although promising results have been achieved in these systems to date, purely reactive systems can still benefit from human intervention. Many purely reactive systems are myopic in their approach: they sacrifice global knowledge for rapid local interaction. Global information can be useful and it is in this capacity that a human operator can interact with a multi-agent control system.

A related problem in teleoperation is that a human operator is potentially overwhelmed by the large amount of data required to control a multi-agent system in a dynamic environment. This phenomenon is referred to as cognitive overload. The approach described in this section provides a mechanism to significantly reduce the human operator's cognitive and perceptual load by allowing the reactive system to deal with each robot's local control concerns. Two principal mechanisms to achieve this are by (1) allowing the operator to act either as a constituent behavior of the society or (2) to allow him/her to supervise the societal behavioral sets and gains, acting only as needed based upon observable progress towards societal task completion.

In this research, the human operator is allowed to control whole societies of agents; not one robot at a time, but rather controlling global behavior for the entire multi-agent system. This is a straightforward extension of our work in both multi-agent robotic systems [1] and teleautonomy [3]. The end product is a simple way for a commander to control large numbers of constituent elements without concern for low-level details (which each of the agents is capable of handling by themselves). In essence, the human

operator is concerned with global social *strategies* for task completion, and is far less involved with the specific behavioral tactics used by any individual agent.

3.1 Forms of Teleoperation

Telerobotic control is facilitated by two methods. The first method allows the human operator to give directional information to the robots. The second method allows the operator to interactively adjust the parameters of the reactive behaviors, thereby changing the overall behavior of the robot society.

3.1.1 Human Operator as a Behavior

The most common form of teleoperation for robots is to give the robot or robots directional information. In our approach, we are concentrating on giving directional information to the society in general, rather than to any particular robot or robots. In addition, the robots do not blindly follow the directional information of the human operator as a remotely controlled vehicle would. Instead, each robot interprets the directional instructions based on its particular situation in the world.

In our initial work, the robots used a motor schema-based reactive control system. For a more detailed account of this approach, see [4]. Each of a robot's primitive behaviors, or schemas, outputs a vector. The direction of the vector indicates the direction that the behavior wants the robot to go in. The magnitude of the vector indicates the amount that the behavior wants the robot to go in that direction. The vectors that are output from all the robots' behaviors are summed and normalized. The robot uses the resulting vector as the direction in which to move.

In our system, the human operator acts as one of the robots' behaviors when giving directional instructions. This is called the teleautonomy behavior. The human operator injects another vector into the system, just as if he were one of the robots' other behaviors. The direction of the vector indicates the direction that the human operator wants the society of robots to move in. The magnitude of the vector indicates the importance that the human operator thinks should be placed on moving in the specified direction. This vector is then summed and normalized along with all the other vectors from the other behaviors, and the resulting vector is used to indicate the direction of movement of the robot.

Thus, the robots do not blindly follow the instructions of the human operator, and each robot interprets the instructions based on his personal situation. For instance, if the human operator instructs the society of robots to move north and there is an obstacle directly north of robot #3, then robot #3 will not collide with the obstacle. The vector from the human operator will point in the direction of the obstacle, but the vector from the robot's **avoid-static-obstacle** behavior will point away from the obstacle. Because these vectors will cancel each other out, robot #3 will not collide

with the obstacle. However, all of the other robots will proceed north, assuming that their particular situation allows them to.

3.1.2 Human Operator as a Behavioral Supervisor

Each of the robots' behaviors has one or more parameters associated with the behavior. The values of these parameters determine exactly how the behavior will function. For instance, one parameter of the **avoid-static-obstacle** behavior is the gain. Increasing the gain for the **avoid-static-obstacle** behavior increases the magnitude of the vector output by the **avoid-static-obstacle** behavior. This has the effect of causing the robot to want to avoid obstacles more.

By adjusting the values of one or more behavioral parameters, the human operator can alter the overall behavior of the robots.

The human operator has the option to adjust the values of individual behavior parameters or to adjust the overall robots' behavior, or personality, in terms of more abstract personality traits, such as Aggressiveness.

First, the human operator can directly manipulate the behavior parameters. Every parameter for all behaviors is represented in the Telop interface. The human operator can adjust a single parameter at a time. This allows a human operator knowledgeable about the behaviors to fine tune the robot society's overall behavior.

The human operator can also manipulate the behavior parameters in terms of abstract personality traits. There is a set of abstract parameters, which are intended to represent general kinds of behavioral or personality adjustments that the human operator might want to make. In our current system the abstract parameters are Aggressiveness, Wanderlust, and Perceptiveness. The value of an abstract parameter controls the value of two or more individual parameters. The three abstract parameters used each control the values of two individual parameters, but an abstract parameter could control considerably more.

The abstract parameter Aggressiveness determines the amount that the robot is focussed on achieving its goal. Increasing the Aggressiveness parameter causes an increase in the **move-to-goal** behavior gain and a decrease in the **avoid-static-obstacle** behavior sphere of influence. The personality effect of this is to cause the robots to focus more on getting to their goal location and worry less about what is in their way. Likewise, decreasing the Aggressiveness parameter causes a decrease in the **move-to-goal** behavior gain and an increase in the **avoid-static-obstacle** behavior sphere of influence. The personality effect of this is to cause the robots to be more careful of obstacles and be less concerned about getting to their goal location.

The abstract parameter Wanderlust determines the desire of the robot to randomly explore the terrain it is in. Wanderlust controls the gain and the persistence of the **noise** behavior. Increasing the Wanderlust causes the robots to move more randomly.

The abstract parameter Perceptiveness determines the distance from the robot beyond which perception of obstacles and other robots is ignored. This is important, because if the robot's perception is noisy at the outer limits, the operator may want the robot to ignore the data coming from that region.

3.2 Teleoperation Interface

The Telop system includes a graphical user interface to facilitate the communication between the human operator and the robots. The interface consists of three parts: the main window, the meta-slider window and the detail-slider window.

3.2.1 Main Window

The Main Window contains an on-screen joystick and some other general controls (see Figure 7).

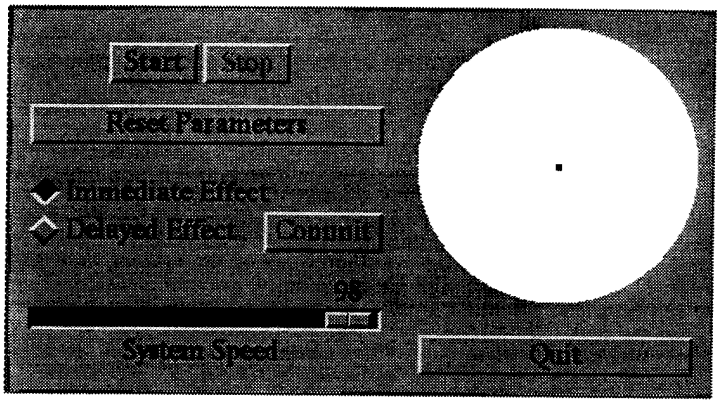


Figure 7: Main Window

The purpose of the on-screen joystick is to provide the human operator a means to give directional information to the robots, as described in the section **Teleoperator as a Behavior**. The position of the joystick indicates the vector that is injected into the system from the teleautonomy behavior. The direction that the joystick is depressed provides the directional component of the vector, and the amount that it is depressed provides the magnitude of the vector.

The main window also contains a toggle button to start and stop the robots from moving, a button to reset all of the behavior parameters to their default values, a slider-bar that controls the speed of the robots, and controls for modes that affect the timing of parameter changes.

The default mode of the system is the Immediate Effect mode. When the system is in this mode, any changes to the parameters, both individual and abstract, are immediately noticed and used by the robots' behaviors.

In the Delayed Effect mode, parameter changes are not immediately noticed by the behaviors until a Commit button is pressed. The behaviors continue using the parameters that the system had when the Delayed Effect mode was entered or the Commit button was last pressed, whichever is most recent. This allows the human operator to change the values of multiple parameters and then have the changes take place simultaneously when the Commit button is pressed.

3.2.2 Meta Window

The Meta Window contains three meta-sliders, and a button that pops up the Detail Window (see Figure 8).

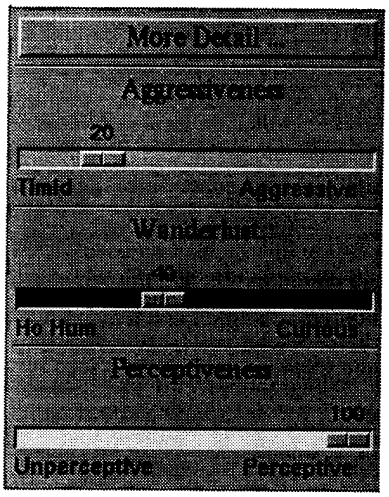


Figure 8: Meta Window

The meta-sliders are slider-bars that control the value of abstract parameters. There is one meta-slider for each of the abstract parameters: Aggressiveness, Wanderlust, and Perceptiveness. We call them meta-sliders because, as the user moves a meta-slider, it then in turn moves two or more detail-sliders, as described below, that correspond to the individual behavior parameters associated with that abstract parameter.

Each meta-slider is a different color. The detail-sliders in the Detail Window that are associated with a meta-slider are the same color.

3.2.3 Detail Window

The Detail Window contains several detail-sliders and a button that removes the detail window (see Figure 9). The detail-sliders are slider-bars, each of which controls the value of an individual behavior parameter.

The detail-sliders are physically arranged into groups. Some groups contain slider-bars for behavior parameters used in a particular state of the robots' task. Other

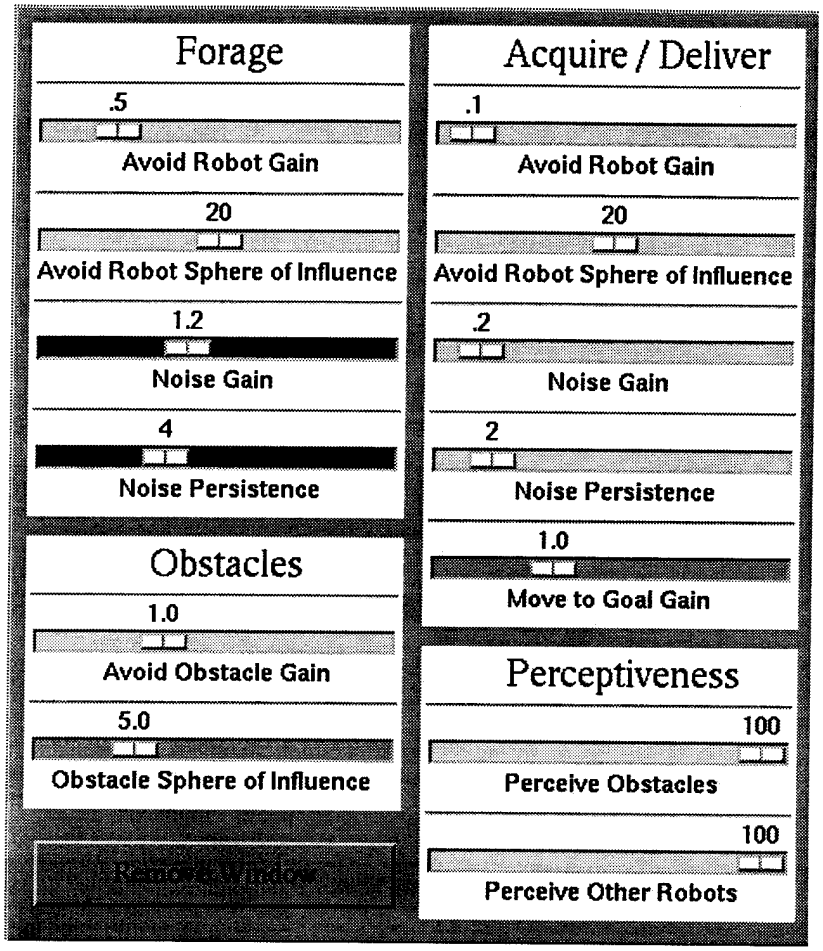


Figure 9: Detail Window

groups contain slider-bars that deal with behavior parameters for a general concept, such as the Obstacle group, which contains slider-bars associated with a behavior for avoiding obstacles.

The groupings that we are using now are based on our own particular task. The slider-bars would either need to be regrouped based on the task they are being used for, or there would have to be a menu of predetermined tasks. Choosing a task from this menu would automatically regroup the slider-bars for that particular task.

Most detail-sliders are the same color, but those detail-sliders that are associated with one of the meta-sliders are the same color as the meta-slider they are associated with.

3.2.4 Usability Tests

A set of usability tests has been conducted on the interface of Telop. These tests yielded useful information for making the interface more helpful and usable. We have made changes to the Telop interface based on the suggestions from these studies.

The evaluators in the usability tests were people working in the robotics group at the Georgia Institute of Technology. In the future, we plan to conduct at least one more set of usability tests on Telop. At that time, we plan to use military students as evaluators.

3.3 Teleautonomy Tests

A set of experiments were done to test the usefulness of the teleautonomy behavior for certain tasks. Only the teleautonomy behavior was tested. The behavioral parameters were not modifyed during the testing. All of the tests were done in a simulation environment.

3.3.1 Simulation Environment

The system is tested on a graphical simulation environment prior to its port to our Denning robots. The objects represented in the simulation environment include robots, obstacles, and attractors. Each robot's trail is depicted by a broken line. Every robot uses the same set of behaviors (a homogeneous society), but the sensory input for each is different, depending on the robot's location within the environment. The robots can sense objects within a certain radius around them. They have the ability to distinguish whether a sensed object is an obstacle, another robot, or an attractor.

The agents have a limited form of communication between themselves. A robot is capable of communicating its current behavioral state or the location of an attractor that it is acquiring or delivering [6]. The communication is simulated by using shared

memory. Each agent only looks at this shared memory when there is no attractor within its sensing range.

In tasks that require the movement of attractors, more than one robot is allowed to contribute to the transport of the object at the same time. The net effect of this cooperation is simulated by having the robots move the attractor farther during each time unit if there are more robots carrying it. The distance traveled while carrying an attractor is determined by the mass of the object and the number of robots carrying it.

3.3.2 Tasks

The use of teleoperation in multi-agent systems was tested for three different tasks. The tasks were foraging, grazing (vacuuming), and herding the robots into a pen. In all three tasks, a teleoperator provided input at his own discretion.

In the foraging task, the robots wander around looking for attractors. When a robot finds a target object, it communicates its location to the other agents while simultaneously moving to acquire it. After its acquisition, the robot carries the attractor back to a home base, then deposits it, and finally returns back to the task of searching for more attractors. If a robot cannot detect an attractor within its sensory radius, it checks to see if any other agent has communicated the location of another candidate goal object. If so, then the robot proceeds to acquire it.

In the grazing task, the robots are placed in an environment studded with obstacles. Initially, all of the floor that is not covered with an obstacle is "ungrazed". Each section of the floor that is ungrazed is treated as if it had an attractor on it. That is, a robot can sense an ungrazed section of floor from a distance, and it can also communicate the presence of an ungrazed section of the floor to the other robots. When an agent passes over an ungrazed region it becomes clean. The task is completed when a certain percentage of the floor, specified in advance, has been grazed. The robots normally wander randomly until an ungrazed floor area is detected.

In the herding task, there is a pen with an opening formed of obstacles in the simulation environment. All the agents are initially outside of the pen. The robots remain in the *forage* state for the duration of the run and wander aimlessly in random directions. The robots are repulsed by the obstacles and the other robots. The task is to get all of the robotic agents inside the pen at the same time.

3.3.3 Results

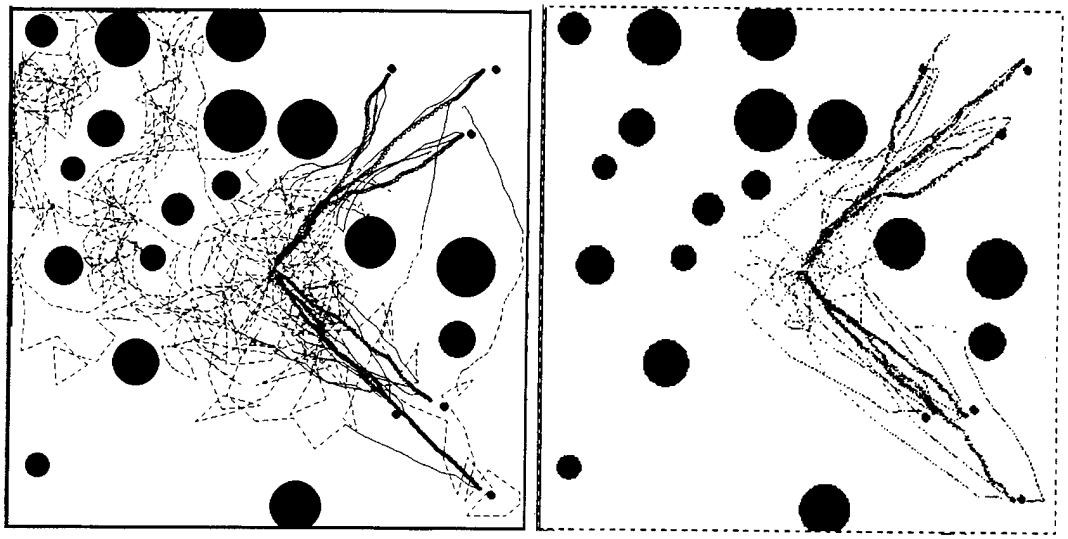
For the foraging and grazing tasks, tests were conducted that compared the total number of steps taken by the robots to complete the tasks with and without the help of a teleoperator. For the herding task, no comparison could be made between teleoperation and no teleoperation, because the likelihood of all the robots wandering

into the pen by themselves at the same time is virtually nil. Interesting information was gained about this task nonetheless.

In the tests conducted for the foraging task, three robots were used to gather six attractors. The density of obstacles in the environment was 10%. The total number of steps required to finish the task was measured both with and without a teleoperator. If teleoperation is used wisely, it can significantly lower the total number of steps required to complete the task by greatly reducing the time spent in the *forage* state (i.e., the number of steps that the robots spend looking for attractors). If none of the agents currently sense an attractor, then the teleoperator can assist by guiding the robots in a fruitful direction. However, once the robots can sense an attractor, the teleoperator should stop giving instructions, unless the instructions are to deal with a particularly troublesome set of obstacles. In general, the robots perform more efficiently by themselves than when under the control of a teleoperator if the agents already have an attractor in sight. The human's instructions tend to hinder the robots if they are already moving to acquire or return an attractor. Indeed, when teleoperation is used at all times, the overall number of steps required for task completion often increases when compared to no teleoperation at all. However, if the human only acts to guide the robots toward an attractor when none are currently detected, significant reductions in time for task completion are possible. The average over several experimental runs of the total number of time steps required for task completion when teleoperation was used in this manner was 67% of the average task completion time when no teleoperation was used.

An example trace of a forage task without teleoperation is shown in Figure 10(a). Another trace of the same forage task with a human teleoperator helping the robots find the attractors when they did not have one in sensing range is shown in Figure 10(b). The robots all started at the home base in the center of the environment. In the run without teleoperation, the robots immediately found the two closer attractors at the lower right. Then they quickly found the two closer attractors at the upper right. At this point, the robots did not immediately detect the remaining two attractors. Two of the three agents proceeded by chance to the left and upper left sides of the environment, wandering unsuccessfully while seeking an attractor. Eventually, the other robot found the attractor in the lower right corner, and the other two robots moved to help with its return. After delivering it to the home base, the robots wandered again for a while without finding the last attractor. Finally, the last attractor was detected and successfully delivered to home base. In the same world with the help of a human teleoperator, the two protracted periods of wandering while searching for attractors are avoided. This indicates the types of environments where the use of teleoperation for the forage task is most beneficial. The greatest benefit from teleoperation can be seen when there are one or more attractors that are far from both the home base and the start locations of the robots. Typically, this is when the robots do not sense the target objects without wandering for a while.

For the grazing task, five robots were used. A sample run of a grazing task is



(a) (b)

Figure 10: Foraging task.(a) Without Teleoperation (b) With Teleoperation

shown in Figure 11. In this case, the robots performed poorly when a large amount of teleoperation was involved. Teleoperation only proved useful when the robots had difficulty in locating a section of ungrazed floor. When the robots had already detected an ungrazed area, they performed better without any input from the teleoperator. The agents' performance degraded considerably, often taking several times longer to complete the task, if teleoperation was used when a robot had already located an ungrazed floor area. Moreover, since remaining untreated areas tend to be clustered together in large patches, the agents typically do not need to spend long periods of time looking for another ungrazed spot (which is opposite the case of the foraging task discussed above). Therefore, the use of teleoperation did not help significantly with the grazing task. When teleoperation was used solely to help the robots find ungrazed floor area when they were not already cleaning, only a 4% improvement in average task completion time performance was observed when compared to not using teleoperation. Thus, when used wisely, teleoperation helped somewhat but not to a large extent.

For the herding task, five robots were herded into a pen that was 36 units long by 18 units wide, with a 12 unit long door in one of the longer sides. All of the robots started at one spot on the side of the pen with the door. In most test runs, the teleoperator encountered no difficulty with this task. He was able to herd the robots into the pen with no problem. In some of the test runs, there were a few minor difficulties, such as robots wandering back out of the pen after having been herded in. However, the teleoperator was still able to complete the task without much frustration and in a reasonable amount of time. The results of a test run for the herding task is shown in Figure 12.

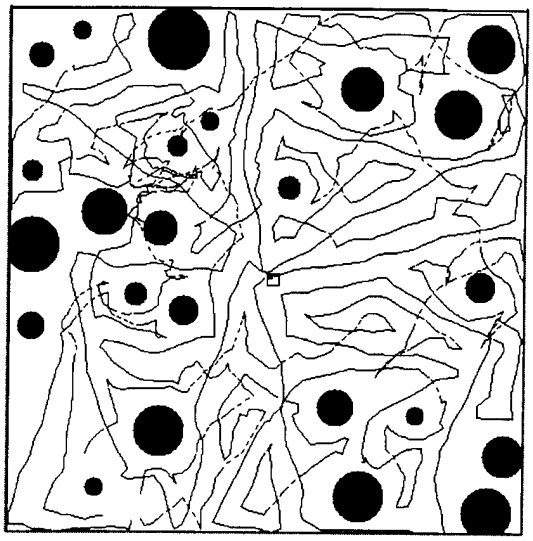


Figure 11: Grazing Task

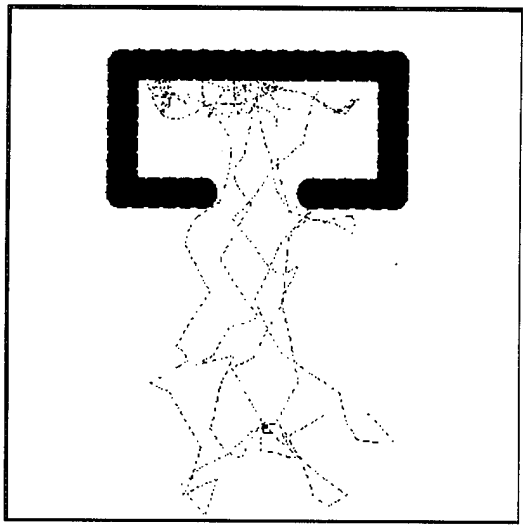


Figure 12: Herding Task

3.3.4 Analysis

Some conclusions can be ascertained from the studies conducted thus far. It should be remembered, however, that these are preliminary studies, and there are many variables that have not yet been explored. For instance, we intend to explore the effects of teleoperation while varying the number of robots for a particular task, to study the role and impact of different inter-agent communication methods on teleoperation, and to conduct an analysis of what types of environments teleoperation is most suited for.

The use of the **teleautonomy** schema in conjunction with the robots' other behaviors proved particularly effective for the foraging task, while being less so for the grazing task (vacuuming). Herding the robots into a pen was arduous but feasible using this method. During foraging, the best results were observed when teleoperation was used only to guide the robots in the direction of an attractor if one had not been previously sensed. For the vacuuming task, teleoperation was not significantly better than no teleoperation, although minor improvements were observed. The best results were again seen when teleoperation was used in guiding the robots towards dirty areas that were outside the sensor (or communication) range of the agents.

Trying to herd the robots into a pen is possible, although a frustrating task. Two conceivable improvements can be used for this task regarding teleoperation. The first is to allow the teleoperator to turn off the input from the teleoperation schema for specific robots but not for others, allowing the operator to concentrate on the outside robots without worrying what effects his actions will have on robots already inside the pen. The other improvement is to allow the teleoperator to completely stop a robot's movement when it is inside the pen. In this way, the output of the teleoperation schema could be thought of as producing a vector that nullifies the vectors produced by the robot's other schemas. However, both of these strategies involve producing different output for the **teleautonomy** schema for different robots. This means that the teleoperator would have a greater burden, defeating the purpose of this research in reducing the cognitive workload.

Another important point is that if the teleoperator is given unrestricted control of the magnitude of the vector produced by the teleoperation schema, it is possible for the teleoperator to force a robot to collide with obstacles and other robots. The teleoperator must be careful when increasing the gain of the **teleautonomy** schema so that this does not occur. It can be a delicate task to override the output of the **noise** schema, which is necessary to cause the robots to quickly move in a particular direction, while not overriding the **avoid-static-obstacle** behaviors.

3.4 Integration with ARPA UGV Project

The Telop system is being developed for use as part of the ARPA UGV project. The integrated demo of Demo C will include the teleautonomy behavior. A tech demo

at Demo C and the integrated demo at Demo II will include both the teleautonomy behavior and the slider bars for modifying behavioral parameters.

3.4.1 Teleautonomy Behavior

We are currently working to make Telop compatible with the DAMN arbiters, which are the arbitration systems used on the vehicles in the UGV project. The main window of the interface has been modified in appearance and functionality to accommodate the differences between the combination mechanism used in our schema system and the arbitration mechanism used in the DAMN arbiters. The joystick now controls the direction and the speed of the society of robots. When the joystick is used, the interface sends messages (via TCX) to the teleautonomy behaviors. These messages indicate the direction and the speed that the society of robots should move in. Two slider bars have been added to adjust the weight that the DAMN turn arbiters and the DAMN speed arbiters use when considering the votes from the teleautonomy behavior. The communication with the DAMN arbiters to change the weights is done using TCX.

We are in the process of implementing the teleautonomy behavior in the same form as the other DAMN behaviors. There are actually two teleautonomy behaviors, one for turning and the other for speed. Each robot will be running one teleautonomy turn behavior and one teleautonomy speed behavior. When the behaviors receive a message from the Telop interface indicating a direction and speed for the society to move in, the behaviors will compute a turn radius and speed based on the situation of the particular robot that the behavior is resident on. This conversion from holonomic to nonholonomic instructions has not yet been implemented. Then the behaviors send their votes for this turn radius and speed to the DAMN arbiter. The votes for turn radii are actually distributed among all the possible turn radii as a gaussian centered at the desired turn radius.

We are also creating an interface for modifying the default parameters that the Telop interface will use when it is started by either MRPL or the STX interface. This default editor will also allow modification of the system defaults for the parameters.

The interfaces are implemented using UIM/X for ease of integration with the other interfaces in the project.

3.4.2 Parameter Modification

For a technical demo at Demo C and for the integrated demo at Demo II, we are implementing detail-sliders and meta-sliders for controlling behavioral parameters both in terms of the individual parameters and abstract personality traits. The detail-sliders will correspond to the possible behaviors running on the vehicles. The grouping of the detail-sliders will be based on the tasks and concepts that are part of the military scenario used in the UGV demos. The meta-sliders will then be set up to control appropriate detail-sliders based on the personality trait that they represent.

There will be two detail-sliders for the weights of each behavior in the arbiters. The first will control the weight in the turn arbiter, and the second will control the weight in the speed arbiter. In addition, we will analyze the internal parameters present in the behaviors used with DAMN. If it is appropriate for a particular parameter, we will include a slider bar in the Telop interface for modifying this parameter.

When a slider bar is moved, the Telop interface will send messages to the appropriate arbiters or behavior on each vehicle. The messages will instruct the arbiter to change the weight that it uses when considering the votes of a particular behavior.

4. Configuration Design Support for Mission Specification

Configuration design tools are being developed which will support graphical construction of abstract configurations which can then be compiled to execute on the MRPL system for UGV Demo II. Planned capabilities include a graphical editor for creating configurations, support for automated configuration evaluation, and a Multi-Robot Programming Language (MRPL) code generator.

4.1 The Configuration Description Language (CDL)

The context free Configuration Description Language (CDL) provides a solid theoretical foundation for specifying architecture and robot independent configurations of behavior-based robots. Societies of robots come in three types; trivial, homogeneous teams, and heterogeneous castes. The language specifies the coordination between members of homogeneous teams, members of heterogeneous castes, assemblages of behaviors for individual robots, as well as perceptual strategies within primitive sensorimotor behaviors.

A preliminary version of the language has been developed. The grammar G generating the language will be described by the notation [10] $G = (V, T, Q, S)$, where V is the set of variables, T is the set of terminal symbols, Q is the set of productions, and S represents the highest-level society (the start variable). Using this notation, G is described as

$$
\begin{aligned}
G \;=\; & (\{S, X, R, A, C, Y, B, P, Z\}, \\
& \{\mathbf{p}_i, \mathbf{m}_j, \mathbf{a}_k, /_l, *_m, +_n, -_o, \%_p, @_q, \#_r, =_s, \{, \}, [,], \langle, \rangle, `, ', (,)\}, \\
& Q, \\
& S)
\end{aligned}
$$

and Q consists of the productions

$$
\begin{array}{lcl}
S & \rightarrow & R \mid \text{`}/_l\ XS\text{'} \mid \text{`}*_m\ \ XS\text{'} \\
X & \rightarrow & XS \mid S \\
R & \rightarrow & \{A\} \\
A & \rightarrow & B \mid [+_n\ YA] \mid [-_o\ YA] \mid [\%_p\ YA] \mid [@_q\ YA] \\
Y & \rightarrow & YA \mid A \\
B & \rightarrow & \langle P\mathbf{m}_j \rangle \mid \langle \mathbf{a}_k P\mathbf{m}_j \rangle \mid \langle P \rangle \mid \langle \mathbf{a}_k P \rangle \\
P & \rightarrow & \mathbf{p}_i \mid (\#_r\ ZP) \mid (=_s\ \ ZP) \\
Z & \rightarrow & ZP \mid P
\end{array}
$$

Where

- S is a society
- X is a list of one or more societies
- R is a single robot
- A is a behavioral assemblage
- Y is a list of one or more assemblages
- B is a primitive sensorimotor behavior
- P is a perceptual module or a coordinated perceptual group
- Z is a list of one or more perceptual modules
- $\mathbf{p}_i, i \in$ *natural numbers* is an instance of a perceptual process
- $\mathbf{m}_j, j \in$ *natural numbers* is an instance of a motor process
- $\mathbf{a}_k, k \in$ *natural numbers* is an instance of an active perception motor process
- $/_l, l \in$ *natural numbers* is an instance of a caste (heterogeneous) society operator
- $*_m, m \in$ *natural numbers* is an instance of a team (homogeneous) society operator
- $+_n, n \in$ *natural numbers* is an instance of an assemblage cooperation operator
- $-_o, o \in$ *natural numbers* is an instance of an assemblage competitive operator
- $\%_p, p \in$ *natural numbers* is an instance of an assemblage sequencing operator
- $@_q, q \in$ *natural numbers* is an instance of the generic assemblage coordination operator
- $\#_r, r \in$ *natural numbers* is an instance of a perceptual fusion operator
- $=_s, s \in$ *natural numbers* is an instance of a perceptual sequencing operator
- ‘ ’ delineates societies
- { } delineates agents (robots)
- [ ] delineates coordinated assemblages
- ⟨ ⟩ delineates primitive sensorimotor behaviors
- ( ) delineates a group of coordinated perceptual modules.

Sensors are explicitly represented to allow parameterization and to facilitate hardware binding. Perceptual modules function as virtual sensors which extract features from one or more sensation streams and generate, as output, a stream of features (individual percepts). Motor modules use one or more feature streams (perceptual inputs) to generate an action stream (a sequence of actions for the robot to perform). Perceptual coordination is the process of linking one or more perceptual modules to motor modules and is partitioned into three categories[2]: sensor fission, action-oriented perceptual fusion, and sensor "fashion". Active perception utilizes a special motor module which generates an action stream to modify the information the sensor is providing. A primitive behavior consists of one or more perceptual modules and a motor module generating a stream of actions based on perceptual inputs. An assemblage can be treated as a single sensorimotor behavior even though it may be recursively composed of many primitive behaviors and coordination strategies. Each individual robot is controlled by a single assemblage.

4.2 Code Generators

The Configuration Network Language (CNL) will be used to represent the output of the CDL compiler. CNL has been designed and implemented as a dataflow language, where the executable functions represented by the nodes are specified using a standard programming language. A compiler generating C++ code for the schema architecture has been developed and is in regular use in the mobile robot lab. A compiler generating MRPL code for the ARPA architecture will be developed this fall.

4.3 MissionLab - Simulation System Implementation

The MissionLab simulation system architecture is shown in Figure 13. The basic parts are a simulation system, a display system, and robot control software. The simulation system is shown in a dotted box.

The simulation system includes a module which can read and interpret overlay files, a database where the scenario information is stored, a module which can read and interpret command files (and construct an internal list of instructions), and a command interpreter which will execute the commands from the internal list of commands. The commands are digested and sent to the individual robots to be executed. This also includes some simple execution monitoring. For details of the simulation system, see [8].

The robot control software is a preconfigured set of assemblages for accomplishing a variety of tasks. Each assemblage is a set of schemas (or low-level behaviors). An executive inside the robot software receives the command from the simulation system, loads information from the command (such as goal locations), and selects the proper set

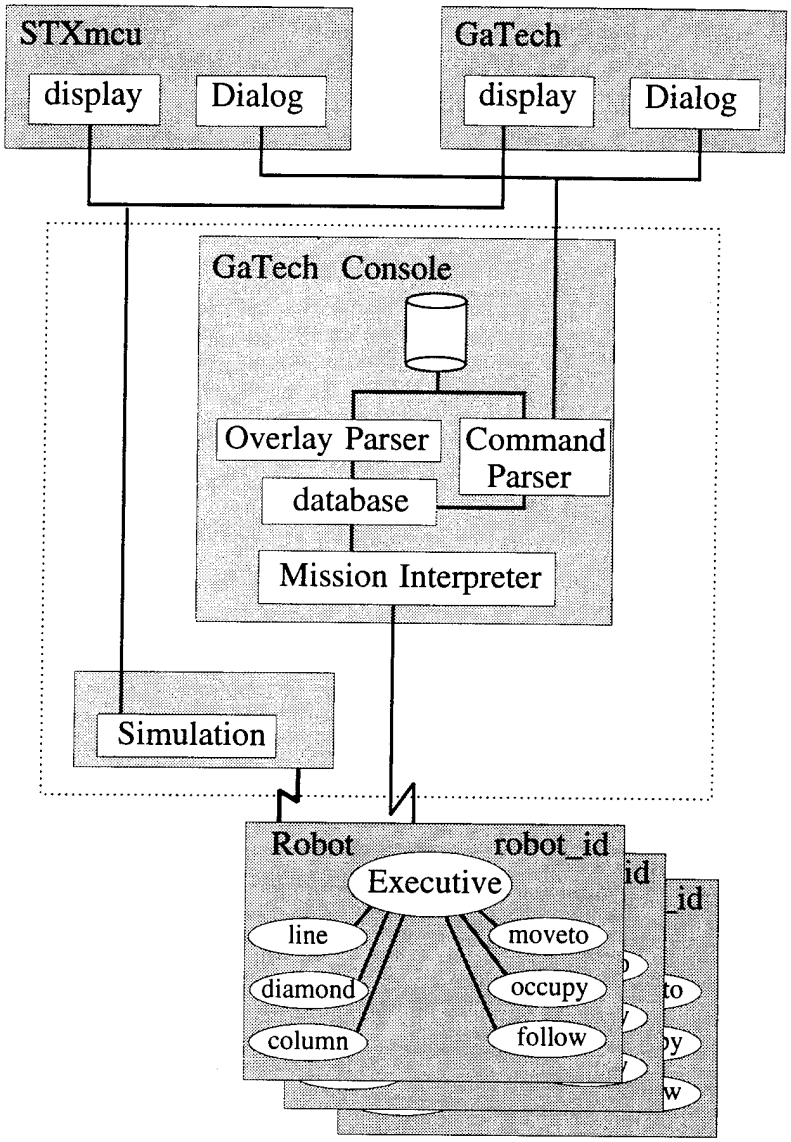


Figure 13: Overview of the current MissionLab system.

of behaviors to execute the desired task. Ultimately, MRPL behaviors will be encoded as an alternate behavior strategy.

The simulation system spawns separate control processes to control the robots. Each of these processes runs a copy of a robot control program. If the robot were real, these processes would communicate with the robot (perhaps via radio links). In our case, the robots are simulated. These separate processes are running independently, in parallel. They communicate with the simulation system using the TCX inter-process communication package [9].

The simulation system also coordinates information exchange between the robots. When commands are executed, the simulation system sends the command to the appropriate robot with whatever modifications are necessary. For instance, a "unit" might consist of several robots. When the unit is commanded to do something, the simulation system uses the console database to determine what robots need to receive the command. The simulation system then sends the command to each robot in the unit. In some cases, the command must be modified slightly before it is sent—such as in a start command so that all the robots in the unit are not positioned on top of each other.

Map overlay information is loaded from overlay description files. This includes information including locations of control measures such as boundaries, assembly areas, lines of departure/lines of contact, attack positions, passage points, gaps, axes of motion, phase lines, battle positions, and objectives. A sample file is shown in Figure 14. Actual map coordinates are not included for simplicity.

```
SCENARIO "Demo-C"
SITE "Demo B site, Colorado"
ORIGIN X Y
CONTROL MEASURES:
Boundary Yankee x1 y1 x2 y2 ... xn yn
LDLC Echo x1 y1  x2 y2  ...  xn yn
AA Alpha x1 y1  x2 y2  x3 y3 ...  xn yn
ATK Bravo x1 y1  x2 y2  x3 y3 ...  xn yn
PP Charlie x y
Gap Delta x1 y1  x2 y2
Axis Foxtrot x1 y1  x2 y2  ...  xn yn
PL Gamma x1 y1  x2 y2  ...  xn yn
BP 1 x1 y1  x2 y2  ...  xn yn
BP 2 x1 y1  x2 y2  ...  xn yn
OBJ Zulu x1 y1  x2 y2  ...  xn yn
```

Figure 14: Example overlay description file without real map coordinates.

4.3.1 Script-based Military Scenario Executive Coordination Operator

A coordination operator has been developed which will interpret a set of steps necessary to accomplish a multiagent mission. Each step is a series of commands to send to the teams of robots involved all at the same time. The step is not complete until all the robots have completed their commands. The tools will load the steps and execute them in our simulation environment with minimal human interaction. Each robot will be configured with a set of behaviors to allow it to execute its commands with appropriate reactive control software.

The robots are configured with a set of behaviors which allow them to perform the necessary tasks (see the lower part of Figure 13). Currently we have behaviors for moving to a specified location, following a line, and occupying a position. Concurrently, the robot can be using any of several formations behaviors, such as line formation, column formation, diamond formation, wedge formation, and no formation. We are also working towards implementation several coordinated movement techniques such as traveling overwatch and bounding overwatch. The executive coordination node in each robot control software receives commands from the simulation system and activates the appropriate behaviors to accomplish the task.

There are two task description files: an overlay description file and a command description file. The overlay description file is a file containing the mission overlay information which was described in the previous section. The command description file contains background information and commands. The background information part includes such things as which overlay file to use, starting positions, and the composition of the units involved. The command information part includes a series of steps to be executed. Each step can be composed of several commands to be executed in parallel.

An example of a command description file is given in Figure 15. Notice that the file references an overlay file. Also notice the readable nature of the command language.

5. Communication in Multiagent Robotic Teams

Multiple cooperating robots are able to complete many tasks more quickly and reliably than one robot alone. Communication between the robots can multiply their capabilities and effectiveness, but to what extent? In this research, the importance of communication in robotic societies is investigated through experiments on both simulated and real robots. Performance was measured for three different types of communication for three different tasks. The levels of communication are progressively more complex and potentially more expensive to implement. For some tasks, communication can significantly improve performance, but for others inter-agent communication is apparently unnecessary. In cases where communication helps, the lowest level of communication is almost as effective as the more complex type. The bulk of these results are derived from thousands of simulations run with randomly generated ini-

```
MISSION NAME "Demo C simulation"
SCENARIO "Demo-C"
OVERLAY test.odl
SP Home 0 0
UNIT Wolf (Wolf-1 ROBOT BFV) (Wolf-2 UGV SSV HUMMER)
COMMAND LIST:
  0. UNIT Wolf START SP-Home 10 0
  1. UNIT Wolf MOVETO AA-Alpha FORMATION diamond traveling-overwatch
 1a. UNIT Wolf OCCUPY AA-Alpha FORMATION Column
  2. UNIT Wolf MOVETO ATK-Bravo FORMATION Column
  3. UNIT Wolf OCCUPY ATK-Bravo FORMATION Diamond
  4. UNIT Wolf MOVETO PP-Charlie FORMATION Column
  5. UNIT Wolf FOLLOW Gap-Delta FORMATION Column
  6. UNIT Wolf FOLLOW Axis-Foxtrot FORMATION Diamond Traveling-Overwatch
     PHASE-LINE PL-Gamma 06-10-94:23:10
  7. UNIT Wolf PASS-PHASE-LINE PL-Gamma
  8. UNIT Wolf-1 MOVETO BP-1 FORMATION Line Bounding-Overwatch AND
     UNIT Wolf-2 MOVETO BP-2 FORMATION Line Bounding-Overwatch
  9. UNIT Wolf-1 OCCUPY BP-1 FORMATION DIAMOND AND
     UNIT Wolf-2 OCCUPY BP-2 FORMATION DIAMOND
 10. UNIT Wolf-1 MOVETO OBJ-Zulu FORMATION Line Bounding-Overwatch AND
     UNIT Wolf-2 MOVETO OBJ-Zulu FORMATION Line Bounding-Overwatch
 11. UNIT Wolf OCCUPY OBJ-Zulu FORMATION Diamond
 12. UNIT Wolf STOP
```

Figure 15: Example Command Description File

tial conditions. The simulation results help determine appropriate parameters for the reactive control system which was ported for tests on Denning mobile robots.

Three different types of communication are evaluated in this research. Using a minimalist philosophy, the first type actually involves no direct communication between the agents. The second type allows for the transmission of state information between agents in a manner similar to that found in display behavior in animals. The third type (goal communication) requires the transmitting agent to recognize and broadcast the location of an attractor when one is located within detectable range.

Our research to date focuses on three tasks: foraging, consuming, and grazing. Foraging consists of searching the environment for objects (referred to as attractors) and carrying them back to a central location. Consuming requires the robot to perform work on the attractors in place, rather than carrying them back. Grazing is similar to lawn mowing; the robot or robot team must adequately cover the environment.

The impact of communication on performance in reactive multiagent robotic systems has been investigated through extensive simulation studies. Performance results for three generic tasks illustrate how task and environment can affect communication payoffs. Initial results from testing on mobile robots are shown to support the simulation studies.

The principal results for these tasks are:

- Communication improves performance significantly in tasks with little environmental communication.
- Communication is not essential in tasks which include implicit communication.
- More complex communication strategies offer little or no benefit over low-level communication.

Future work in this area is concerned with societal performance in fault-tolerant multiagent robotic systems; where unreliable communication may be present and the robotic agents have the potential for failure.

REFERENCES

[1] Arkin, R.C., "Cooperation without Communication: Multi-agent Schema Based Robot Navigation", *Journal of Robotic Systems*, Vol. 9(3), April 1992, pp. 351-364.

[2] Arkin, R.C., "The Multiple Dimensions of Action-Oriented Perception: Fission, Fusion, Fashion", Working notes of *AAAI 1991 Fall Symposium on Sensory Aspects of Robotic Intelligence*, Monterey, CA, Nov. 15-17, 1991.

[3] Arkin, R.C., "Reactive Control as a Substrate for Telerobotic Systems", *IEEE Aerospace and Electronics Systems Magazine*, Vol. 6, No. 6, June 1991, pp. 24-31.

[4] Arkin, R.C., "Motor Schema-Based Mobile Robot Navigation", *International Journal of Robotics Research*, Vol. 8, No. 4, August 1989, pp. 92-112.

[5] Arkin, R.C. and Ali, K.S., "Integration of Reactive and Telerobotic Control in Multi-agent Robotic Systems", *From animals to animats 4: Proc. 4th International Conference on Simulation of Adaptive Behavior*, Brighton, England, Aug. 1994, pp. 473-478

[6] Arkin, R.C., Balch, T., and Nitz, E., "Communication of Behavioral State in Multi-agent Retrieval Tasks", *Proc. 1993 IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993, Vol. 3, pp. 588-594.

[7] Brooks, R., Maes, P., Mataric, M., and More, G., "Lunar Base Construction Robots", *IEEE International Workshop on Intelligent Robots and Systems (IROS '90)*, pp. 389-392, Tsuchiura, Japan, 1990.

[8] Cameron, J.M. and MacKenzie, D.C., "Specifying complex military scenarios", Georgia Institute of Technology, *Working paper, contact authors.*

[9] Fedor, C., "TCX: Task Communication," (User manual for TCX, available through the Robotics Institute), Carnegie Mellon University, Feb. 15, 1993.

[10] Hopcroft, J.E. and Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, pp. 79, 1979.

[11] Mataric, M., "Minimizing Complexity in Controlling a Mobile Robot Population", *1992 IEEE International Conference on Robotics and Automation* , Nice, pp. 830-835.

6. Publications to Date Resulting from this Research

- FORMATION BEHAVIORS

  1. Balch, Tucker, 1994 "Motor schema-based formation control for multi-agent robot teams", *Working paper, contact author.*

- TELEAUTONOMOUS CONTROL

  1. Arkin, R.C. and Ali, K., 1994. "Integration of reactive and telerobotic control in multi-agent robotic systems", *Proc. Third International Conference on Simulation of Adaptive Behavior, (SAB94) [From Animals to Animats]*, Brighton, England, Aug. 1994, pp. 473-478.
  2. Ali, Khaled S., 1994 "Telop: Teleoperation of multi-agent reactive robotic systems", *Working paper, contact author.*

- MISSION SPECIFICATION

  1. MacKenzie, D. and Arkin, R.C., 1993. "Formal specification for behavior-based mobile robots", *Mobile Robots VIII*, Boston, MA, Nov. 1993, pp. 94-104.
  2. MacKenzie, Douglas C., 1994 "A Design Methodology for the Configuration of Behavior-Based Mobile Robots", *Ph.D. Thesis Proposal*, 1994.
  3. Cameron, Jonathan M. and MacKenzie, Douglas C., 1994 "Specifying complex military scenarios", *Working paper, contact authors.*

- INTER-ROBOT COMMUNICATION

  1. Balch, T. and Arkin, R.C., 1994. "Communication in reactive multiagent robotic systems", to appear in *Autonomous Robots*, Vol. 1, No. 1.